Representation Compensation Networks for Continual Semantic Segmentation

XXX

摘要

针对持续性地训练细胞核分割模型时可能存在的隐私和存储问题，在 Representation Compensation Networks for Continual Semantic Segmentation(RCIL) 的工作之上，我们提出了一种基于多类型细胞核病理图像的持续语义分割新方法。首先，为了平衡模型对不同旧类知识的记忆能力，通过在经典输出级知识蒸馏函数中增加一个调制项，我们提出了一个新的知识蒸馏函数，称为焦点知识蒸馏。其次，通过将自适应权重加权因子应用于所提出的焦点知识蒸馏，保持了模型记忆旧知识的稳定性和学习新类的可塑性。第三，设计了一个新的的特征级知识蒸馏，进一步缓解了模型对旧知识的遗忘。最后，我们进行了严格的消融研究和对比实验，证明了我们所提出的方法的有效性。

关键词：灾难性遗忘；持续学习；语义分割；细胞核分割

1 引言

细胞核分割是语义分割的难点之一，对肿瘤的诊断和预后具有重要意义。近年来，基于卷积神经网络 (Convolutional Neural Networks, CNNs) 的全监督细胞核分割模型取得了显著的进步[1][2][3]，在一定程度上促进了计算机辅助诊断的发展。尽管如此，这些方法中的大多数都要求在训练过程中所有的图像和注释都必须可用，这使得它们在没有旧类的注释的数据集上训练时失去了区分旧类的能力 (灾难性遗忘)。一种常见的方法是重建包含所有注释的数据，并用它重新训练模型，称为联合训练。然而，重建包含所有注释的数据集的成本是昂贵的，因为 (1) 细胞的高密度性使得注释工作花需要花费大量的时间，(2) 只有具有病理知识的专家才能对数据进行注释，这使得注释工作难以轻易进行，(3) 对先前图像数据的访问受到患者隐私的限制，阻碍了细胞核图像的获取。这些问题导致在没有任何旧类注释的情况下训练细胞核分割模型时，所获得的模型对旧类的识别性能较差 (灾难性遗忘) 的巨大挑战。

持续学习作为一种在无限数据流中不断学习的学习模式，正是为了解决上述挑战而被学者广泛研究。基于持续学习的语义分割称为持续语义分割（CCS）。在 CCS 场景中，给定在之前的训练集上训练好的模型和只包含新类注释的新数据集，目标是克服灾难性遗忘，使模型能够很好地区分所有学习过的类，包括旧类和新类。在这项工作中，基于 Representation Compensation Networks for Continual Semantic Segmentation（RCIL）[4]，我们提出使用并行 Unet 结构来有效分割细胞核。受到 RCIL 中池化集合蒸馏的启发，我们在中间层引入了一种功能更强大的池化知识蒸馏，命名为 Eight Connected Pooled Distillation。它可以聚合局部不重叠的邻域信息以及获得全局范围的决策关系，进一步缓解灾难性遗忘。在实验中，我们发现当旧类数量不平衡时，简单的使用输出级的知识蒸馏会导致模型更加容易忘记数量少的旧类，因此我们提出了一个平衡记忆旧类知识的函数，命名为 Focal Knowledge Distillaion。我们的方法和 RCIL 已经有了大不同，整体框架图如图 1 所示。

2 相关工作

2.1 多型细胞核分割

不同类型的细胞在癌症的发生、发展和临床结果中起着至关重要的作用，准确分割其细胞核是计算机辅助诊断和肿瘤微环境分析的前置任务。传统技术主要基于背景减法和颜色阈值，但对阈值选择的依赖和复杂的后处理使其无法处理具有挑战性的情况。随着 CNN 的发展，深度学习模型已广泛应用于核分割，在全监督设置下，所有类的图像和注释同时可用，这取得了令人满意的性能。而问题的关键在于构建一个高质量的多类型细胞核数据集是非常昂贵的，因为收集和注释都需要领域专家进行繁琐的工作，并且要遵守严格的隐私法规。此外，随着癌症不同阶段的研究，分割的目标也发生了变化，在一段时间内只能获得部分细胞类型的数据，这需要模型随着时间的推移不断更新，逐渐增加类别。由于上述核分割模型存在灾难性遗忘，在学习新信息时容易完全、突然地忘记之前学过的知识，因此迫切需要开发可以用于增量类训练的方法。

2.2 持续学习类增强

类增量持续学习旨在消除灾难性遗忘，在图像分类任务中得到了广泛的研究。目前大多数的方法都是基于这些思想: 动态架构、重放、惩罚计算和知识蒸馏。基于动态架构的方法自适应地扩展其网络结构，以适应来自新类的最新知识。基于重放的方法利用一组来自旧类的训练数据，如原始 (可以分类为一个单独的类别称为排练) 或生成的图像。基于惩罚计算的 methods 保护模型内部的重要权重，防止遗忘。基于知识蒸馏的方法利用旧模型，通过约束新模型在输出或特征上的变化，将关于旧类的知识转移到新模型中，是迄今为止被广泛应用的主流，也是本文重点关注的。在自然图像和医学图像的语义分割中，基于上述思想或其组合的方法都取得了令人印象深刻的效果，而最近也提出了一些有针对性的方法。MiB[5]通过应用交叉熵的无偏版本和输出级知识蒸馏损失来解决背景偏移问题，并为分类器提供无偏权重初始化规则。ILT[6]引入了特征级知识蒸馏，通过最小化新旧模型特征的 L2 距离，再加上冻结编码器的权重。PLOP[7]提出通过局部 POD 蒸馏来保留短期和长期空间依赖性，以防止灾难性遗忘。在这些工作的基础上，我们进一步探索了输出级和特征级知识蒸馏的有效性。

3 本文方法

3.1 问题定义

在标准语义分割设置中，设 $D = (X, Y)$ 表示数据集，其中 X 和 Y 分别表示输入空间和输出空间。对于每一对 (x_i, y_i)，$x_i \in X$ 表示输入图像，$y_i \in Y$ 表示对应的分割 ground-truth。标准语义分割任务的目标是训练一个特征提取器 $f(\cdot; \Phi) \in \mathbb{R}^k$ 和一个分类 $\mathbf{h} \in \mathbb{R}^{k \times |\mathcal{Y}|}$，其中 $|\mathcal{Y}|$ 是标签集 Y 的基数。我们通常通过最小化交叉熵损失来训练 M，

$$\mathcal{L}_{ce}(x; \theta) = -\frac{1}{N} \sum_{i=1}^{N} \log \hat{p}^{(y_i)}(x_i), \tag{1}$$

其中：

$$\hat{p}(x) = \sigma(h^T f(x; \Phi)), \tag{2}$$

$\sigma(\cdot)$ 是 softmax 函数。

在持续语义分割场景中，我们通过多个步骤持续地训练模型。训练集 D 由多个子集组成，表示

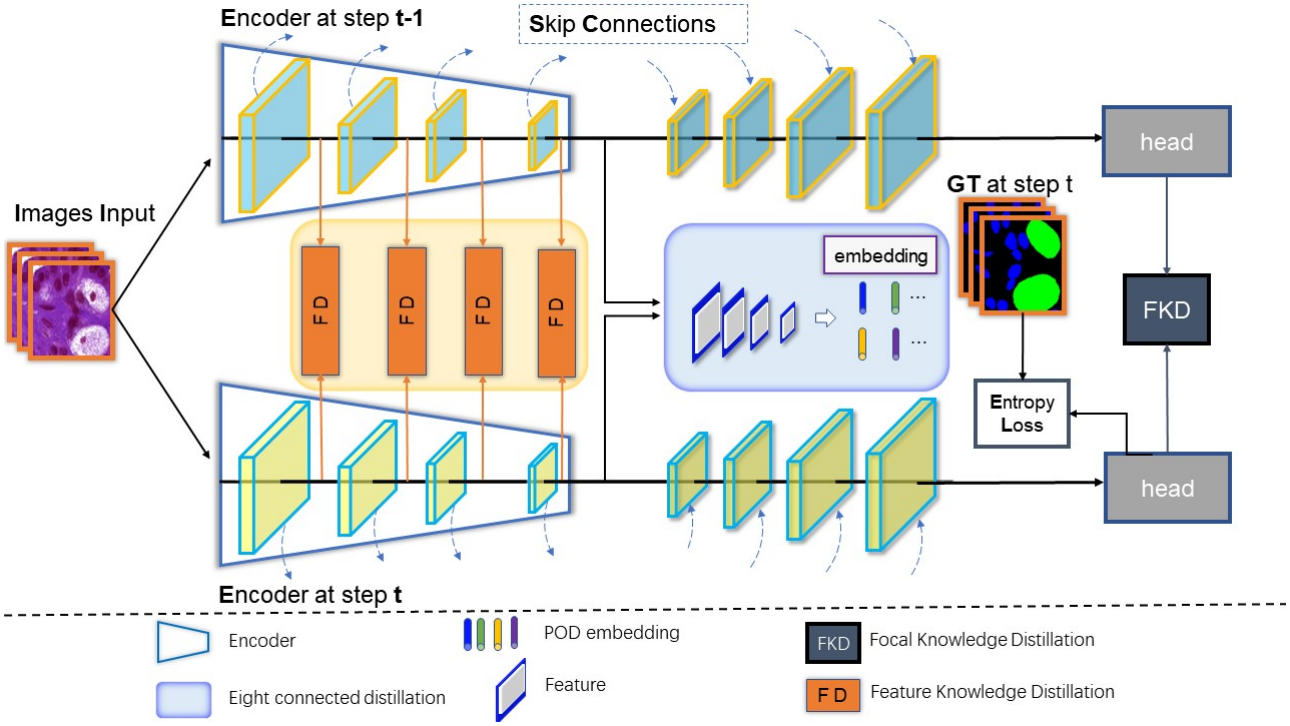

图 1: 方法框架图

为 $\{\mathcal{D}_0, \mathcal{D}_1 ... \mathcal{D}_{t-1}\}$。对于每一步 t，训练集 D_t 只包含新添加的标签 C_t，而旧的标签 $\mathcal{C}_{0...t-1}$ 通常被视为背景。给定数据集 $\mathcal{D}_t$，我们的目标是实现一个具有参数 θ_t 的模型 $\mathcal{M}_t$，它可以在所有见过的标签 $\sum_{k=0}^{t} \mathcal{C}_k$ 上表现良好。为了减轻灾难性遗忘，通常的做法是在式 (1) 中加入特征蒸馏项。然后，通过优化以下目标函数对模型 $\mathcal{M}_t$ 进行更新:

$$\mathcal{L}(\theta_t)_{total} = l_{ce}(x; \theta_t) + \alpha l_{kd}(x; \theta_t), \tag{3}$$

其中 α 是平衡损失项重要性的超参数。$l_{kd}(x; \theta_t)$ 是常用地特征蒸馏，被添加到特征提取器 $f(\cdot; \Phi_t)$ 以保留旧类的知识:

$$l_{kd}(x; \theta_t) = \|f(x; \theta_t) - f(x; \theta_{t-1})\|^2, \tag{4}$$

式中 $|\cdot|$ 表示欧氏距离。

3.2 Focal knowledge distillaion

我们从经典的输出级知识蒸馏方法[8]入手:

$$l_{ckd}(x; \theta_t) = -\frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \sum_{c \in \mathcal{Y}^{t-1}} P_x^{t-1}(i, c) \log \hat{P}^t(i, c), \tag{5}$$

其中，$\hat{P}_x^t(i, c)$ 表示为像素 i 的类别 c 的概率，由 $\mathcal{M}_t$ 给出，但在 $\mathcal{Y}_{t-1}$ 中的所有类别上重新归一化。

$$\hat{P}_x(i, c) = \begin{cases} 0 & if c \neq b \\ P_x^t(i, c) / \sum_{k \in \mathcal{Y}^{t-1}} P_x^t(i, k) & if c = b, \end{cases} \tag{6}$$

其中“c=b”表示旧类被视为背景。为便于下式书写，式中的 $-\frac{1}{|\mathcal{I}|} \sum_{i \in \mathcal{I}} \sum_{c \in \mathcal{Y}^{t-1}}$ 记为 $S_i S_c$。而我们提出的 focal knowledge distillation 定义如下：

$$l_{fkd}(x; \theta_t) = \begin{cases} 0 & if c \neq b \\ S_i S_c m_f P_x^{t-1}(i, c) \log \hat{P}^t(i, c) & if c = b. \end{cases} \tag{7}$$

为便于书写，我们将调制因子 $(\hat{P}_x^{t-1}(i,c) - \hat{P}_x^t(i,c))^\gamma$ 表示为 m_f。我们的知识蒸馏函数有两个主要特点。(1) 当一个旧类 c 容易被遗忘时，那么 $\hat{P}_x^t(i,c)$ 会很小，调制因子接近于 $\hat{P}_x^{t-1}(i,c)$，损失不受影响。当 $\hat{P}_x^t(i,c) \to \hat{P}_x^{t-1}(i,c)$ 时，因子趋近于 0，易被记忆的类的损失降低。(2) 聚焦参数 γ 平滑地调整了记忆速率。

3.3 Eight connected distillation with Local POD

为了进一步缓解对已有知识的遗忘，我们还提出了我们的特征级蒸馏，称为八连池化蒸馏。如图 2(a) 所示，PLOP[7]在水平方向和垂直方向上分别简单地利用了条带池化。如图 2(b) 所示，我们在每个方向 (八个相连方向) 上应用条带池化。

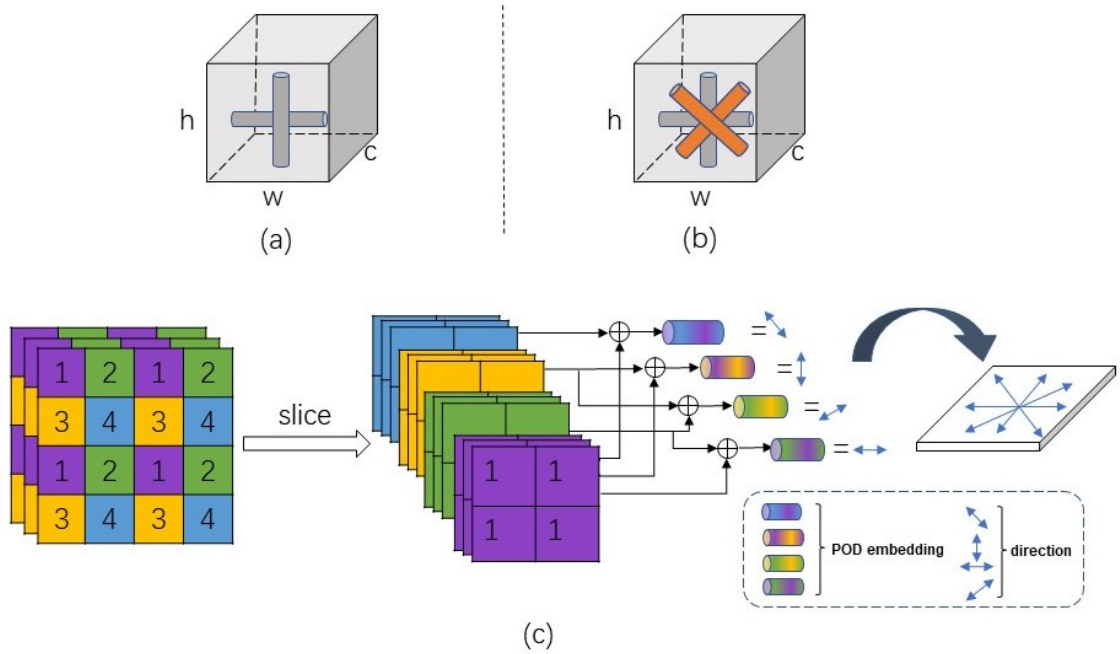

图 2: Eight connected Pooled Cube

如图 2（c）所示，给定特征图 $X_i \in \mathbb{R}^{1\times H\times W\times C}$，我们每隔一个像素取一个值，然后将一个特征划分为四个互补的子特征。我们将这四个子特征表示为 $\{X_{i_1}, X_{i_2}, X_{i_3}, X_{i_4}\}$：

$$\begin{aligned} X_{i_1} &= X_i[:, 0::2, 0::2, :] \\ X_{i_2} &= X_i[:, 0::2, 1::2, :] \\ X_{i_3} &= X_i[:, 1::2, 0::2, :] \\ X_{i_4} &= X_i[:, 1::2, 1::2, :]. \end{aligned} \tag{8}$$

为了获得八连通数据，我们将 X_{i_1} 与 X_{i_2}, X_{i_1} 与 X_{i_3}, X_{i_1} 与 X_{i_4}, X_{i_2} 与 X_{i_3} 相加，分别得到水平方向数据 X_{i_h}、垂直方向数据 X_{i_v}、对角线方向数据 X_{i_d}、次对角线方向数据 X_{i_s}:

$$\begin{aligned} X_{i_h} &= X_{i_1} + X_{i_2} \\ X_{i_v} &= X_{i_1} + X_{i_3} \\ X_{i_d} &= X_{i_1} + X_{i_4} \\ X_{i_s} &= X_{i_2} + X_{i_3} \\ X_I &= \left[X_{i_h} \| X_{i_v} \| X_{i_d} \| X_{i_s}\right], \end{aligned} \tag{9}$$

其中 $\left[\cdot \| \cdot\right]$ 表示将数据沿着 0 维拼接。然后我们对 X_I 使用 $H \times C$ 和 $W \times C$ 的带状池化，最后将结

果拼接在一起，从而获得 POD embedding $\Phi \in \mathbb{R}^{4\times(H+W)\times C}$:

$$\Phi(X_I) = \Big[\frac{1}{W}\sum_{w=1}^{W} X_I[:,:,w,:] \Big\| \frac{1}{H}\sum_{h=1}^{H} X_I[:,h,:,:]\Big], \tag{10}$$

然后，POD 损失定义为最小化当前网络上两组 embedding 之间的 L2 距离:

$$\mathcal{L}_{POD}(\theta_t) = \Big\|\frac{1}{L}\sum_{l=1}^{L} \Phi(f^l_{\theta_t}(X_I)) - \Phi(f^l_{\theta_{t-1}}(X_I))\Big\|^2 \tag{11}$$

4 复现细节

4.1 Focal knowledge distillaion

涉及论文 idea 实现，暂不开源，请谅解。

4.2 Eight connected pooled distillaion

涉及论文 idea 实现，暂不开源，请谅解。

4.3 并行 Unet

见附页图 3 和图 4 源码所示。

5 实验结果分析

在本节中，我们将分析我们提出的 Focal Knowledge distillation 和 Eight connected pooled distillation 在持续语义分割中的有效性。

Focal knowledge distillation 我们在 MoNuSAC-2020 上进行所有的消融实验，并选择 [4] 中介绍的表征补偿模块 (RC) 作为我们的基线，值得注意的是在合并两个并行卷积分支后，我们并没有冻结参数，因为这会导致分割性能变差。如表 1 所示。我们提出的焦点知识蒸馏，极大地缓解了旧类尤其是数量稀少的旧类的遗忘。

	MoNuSAC 3-1 (2 tasks)					MoNuSAC 2-1 (3 tasks)				
	disjoint					disjoint				
Method	*Neutrophils*	*Macrophages*	*Epithelial*	*Lymphocyte*	*Average*	*Neutrophils*	*Macrophages*	*Epithelial*	*Lymphocyte*	*Average*
KD	0.458	0.433	0.630	0.621	0.536	0.394	0.450	0.603	0.673	0.530
FKD	**0.481**	0.444	0.633	**0.647**	**0.551**	**0.448**	0.446	0.609	0.671	**0.544**

表 1: Comparison studies of focal knowledge distillation, with task 3-1 and task 2-1 on the imbalanced dataset MoNuSAC.

Eight Connected Distillation 如表 2 所示，在基线 RC(仅表征补偿模块) 的设置下，我们研究了八连通池化蒸馏的有效性。multi-scale Pooled Cube 知识蒸馏 [4] 进一步缓解了旧类的灾难性遗忘，但效果不如其他方法。事实上，multi-scale Pooled Cube 包含了过多的重叠区域，导致许多的无效计算。文献 [7] 中引入的 multi-scale local strip 知识蒸馏大大优于 Pooled Cube，说明 multi-scale local strip 可以有效聚合短距离和长距离的决策关系。我们的方法比基线和第二名分别高出约 19.71% 和 2.84%。我们将成功归因于以下两个原因:(1) 通过对特征进行相同的带状池化操作，我们的方法继承了 multi-scale local strip 所具备的汇聚短距离和长距离依赖关系的能力，(2) 通过组合切片，我们在每个方向上构造

RC	Strip[7]	Pooled Cube[4]	EC-Strip	3-1
✓				48.61
✓	✓			65.48
✓		✓		61.14
✓			✓	68.32

表 2: The final mIoU(%) of ablation study about Eight Connected Distillation. Experiments are conducted on 3-1 overlapped setting on MoNuSAC.

一组数据，使像素从不重叠的邻居中获得更多决策关系。

6 总结与展望

针对持续性地训练细胞核分割模型时可能存在的隐私和存储问题，在 Representation Compensation Networks for Continual Semantic Segmentation(RCIL) 的工作之上，我们提出了一种基于多类型细胞核病理图像的持续语义分割新方法。首先，为了平衡模型对不同旧类知识的记忆能力，通过在经典输出级知识蒸馏函数中增加一个调制项，我们提出了一个新的知识蒸馏函数，称为焦点知识蒸馏。其次，通过将自适应权重加权因子应用于所提出的焦点知识蒸馏，保持了模型记忆旧知识的稳定性和学习新类的可塑性。第三，设计了一个新的的特征级知识蒸馏，进一步缓解了模型对旧知识的遗忘。最后，我们进行了严格的消融研究和对比实验，证明了我们所提出的方法的有效性。

未来的研究包括将我们的方法应用于其他持续学习场景，如类增量开放自然数据集和域增量语义分割，此外，在更多的病理学数据集上测试我们的方法，并将有效的成果集成到合适的肿瘤微环境分析系统中。

参考文献

[1] CHEN Y, LI X, HU K, et al. Nuclei segmentation in histopathology images using rotation equivariant and multi-level feature aggregation neural network[C]//2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). 2020: 549-554.

[2] LAL S, DAS D, ALABHYA K, et al. NucleiSegNet: robust deep learning architecture for the nuclei segmentation of liver cancer histopathology images[J]. Computers in Biology and Medicine, 2021, 128: 104075.

[3] ZHAO B, CHEN X, LI Z, et al. Triple U-net: Hematoxylin-aware nuclei segmentation with progressive dense feature aggregation[J]. Medical Image Analysis, 2020, 65: 101786.

[4] ZHANG C B, XIAO J W, LIU X, et al. Representation compensation networks for continual semantic segmentation[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2022: 7053-7064.

[5] CERMELLI F, MANCINI M, BULO S R, et al. Modeling the background for incremental learning in semantic segmentation[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020: 9233-9242.

[6] MICHIELI U, ZANUTTIGH P. Incremental learning techniques for semantic segmentation[C]//

Proceedings of the IEEE/CVF international conference on computer vision workshops. 2019: 0–0.

[7] DOUILLARD A, CHEN Y, DAPOGNY A, et al. Plop: Learning without forgetting for continual semantic segmentation[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2021: 4040-4050.

[8] HINTON G, VINYALS O, DEAN J. Distilling the knowledge in a neural network[J]. arXiv preprint arXiv:1503.02531, 2015.

7 附加内容

```
class Unet(nn.Module):
    def __init__(self,
                 in_channels,
                 filters,
                 norm_act=nn.BatchNorm2d,
                 ):

        super(Unet, self).__init__()


        self.u_conv1 = nn.Conv2d(in_channels=in_channels, out_channels=filters[3],
kernel_size=1)
        self.u_conv2 = nn.Conv2d(in_channels=filters[1], out_channels=filters[0],
kernel_size=1)

        self.u_decoder4 = DecoderBlock(filters[3], filters[2],norm_act=norm_act)
        self.u_decoder3 = DecoderBlock(filters[2], filters[1],norm_act=norm_act)
        self.u_decoder2 = DecoderBlock(filters[0], filters[0],norm_act=norm_act)
        self.u_decoder1 = DecoderBlock(filters[0], filters[0],norm_act=norm_act)

        self.u_conv1_new = nn.Conv2d(in_channels=in_channels, out_channels=filters[3],
kernel_size=1)
        self.u_conv2_new = nn.Conv2d(in_channels=filters[1], out_channels=filters[0],
kernel_size=1)

        self.u_decoder4_new = DecoderBlock(filters[3], filters[2], norm_act=norm_act)
        self.u_decoder3_new = DecoderBlock(filters[2], filters[1], norm_act=norm_act)
```

图 3: 并行 Unet 实现

```
        self.u_decoder2_new = DecoderBlock(filters[0], filters[0], norm_act=norm_act)
        self.u_decoder1_new = DecoderBlock(filters[0], filters[0], norm_act=norm_act)

    def forward(self, x):
        # print(x[0].shape)
        # print(x[1].shape)
        # print(x[2].shape)
        # print(x[3].shape)
        # print(x[4].shape)
        # Decoder
        e = functional.leaky_relu_(self.u_conv1(x[4]),negative_slope=0.01)
        d4 = self.u_decoder4(e+x[3]) + x[2]
        d3 = self.u_decoder3(d4) + x[1]
        e = functional.leaky_relu_(self.u_conv2(d3),negative_slope=0.01)
```

图 4: 并行 Unet 实现-续